HPC: HW5 Report

2017-19841 최창민

병렬화 방식, 성능최적화, 메인 matmul

행렬곱 Kernel

행렬곱 커널은 과제 4에서 구현한 커널을 거의 그대로 활용했습니다. TS x TS모양의 thread block을 가지고 동시에 shared memory로 값을 불러온 다음 연산을 하였습니다.

```
__global__ void matmul_cal(const float *A, const float *B, float *C, int M, int N, int
K) {
  const int col = threadIdx.x;
  const int row = threadIdx.y;

  const int global_col = TS * blockIdx.x + col;
  const int global_row = TS * blockIdx.y + row;

  __shared__ float Asub[TS][TS];
  __shared__ float Bsub[TS][TS];

  float c = 0.0;
  const int numTiles = K / TS;
  for (int t = 0; t < numTiles; ++t) {
    const int tiledRow = TS * t + row;
    const int tiledCol = TS * t + col;
    Asub[row][col] = A[tiledCol + K * global_row];
    Bsub[row][col] = B[global_col + N * tiledRow];

    __syncthreads();

    for(int k = 0; k < TS; k++) {
      c += Asub[row][k] * Bsub[k][col];
    }

    __syncthreads();
  }
  C[global_col + N * global_row] = c;
}
```

메인 matmul함수

matmul함수에서는 노드간 데이터통신, GPU-Host간 데이터통신, Kernel연산을 겹쳐보려고 노력해봤습니다. A를 M에 대해서 잘라서 분배하는 것을 시도했는데, 노드간 통신헤서 scatter, gather, send, recv등 다양한 것을 써봤지만, 너무 자주 호출하는 경우 오버해드가 많이 커졌습니다. GPU-Host간 통신의 경우 버퍼링을 써서 연산과 중첩했습니다.

CUDA와 OpenCL, 함수에서 사용하는 API

간단하게 느낀 점은, OpenCL은 커널함수만 잘 짜면 메인 matmul함수는 비교적 쉽게 작성할 수 있었던 것 같습니다. 그래서 OpenCL은 쉽게 띄울 수 있는 것이 장점인 것 같고, CUDA는 그에 비해 작성하는 것이 복잡하지만 좀 더 섬세하게 컨트롤 할 수 있는 것이 장점입니다.또, OpenCL은 Nvidia가 아닌 GPU에서도 사용할 수 있는 것 또한 장점입니다.

API의 경우 주로 Malloc과 Free, Create/Destory Event/Stream을 썼습니다.